SOFTWARE MAINTENANCE: RESEARCH AND PRACTICE
J. Softw. Maint: Res. Pract. **10**, 249–278 (1998)

Research

Q-MOPP: Qualitative Evaluation of Maintenance Organizations, Processes and Products

LIONEL BRIAND[1,*], YONG-MI KIM[2], WALCÉLIO MELO[3], CAROLYN SEAMAN[4] and VICTOR R. BASILI[4]

[1]*Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany*
[2]*Q-Labs, Inc., 4511 Knox Road #305, College Park MD 20740–3380, U.S.A.*
[3]*Oracle Consulting Services, Oracle do Brasil Sistemas Ltda., SCN Q.2 Bl. A S. 604, Brasília D.F., 70712-900, Brazil*
[4]*Experimental Software Engineering Group, University of Maryland, College Park MD 20742, U.S.A.*

SUMMARY

In this paper, we propose a qualitative, inductive method for characterizing and evaluating software maintenance processes, thereby identifying their specific problems and needs. This method encompasses a set of procedures which attempt to determine causal links between maintenance problems and flaws in the maintenance organization and process. This allows for a set of concrete steps to be taken for maintenance quality and productivity improvement, based on a tangible understanding of the relevant maintenance issues in a particular maintenance environment. Moreover, this understanding provides a solid basis on which to define relevant software maintenance models and measures. A case study of the application of this method, called Q-MOPP, is presented to further illustrate its feasibility and benefits. © 1998 John Wiley & Sons, Ltd.

KEY WORDS: maintenance processes; defect causality; actor–dependency modelling; maintenance management; process modelling; product modelling

1. INTRODUCTION

In recent years the definition and improvement of software processes has played an increasingly prominent role in software development and maintenance. The improvement of software maintenance processes is of particular interest because of the length of time spent in maintenance during the software life cycle, and the ensuing lifetime costs, as well as the large number of legacy systems still being maintained. Maintenance improvement, in a given organization, requires building an understanding of what is actually happening in its maintenance projects (the term 'project' in this paper refers to the continuous mainte-

* Correspondence to: Lionel Briand, Fraunhofer IESE, Kaiserslautern, Germany. Email: briand@iese.fhg.de

Contract/grant sponsor: NASA; Contract/grant number: NSG-5123

CCC 1040-550X/98/040249–30$17.50

Received 14 October 1997
Revised 8 January 1998

nance of a given system), in conjunction with building a measurement program to monitor and assess improvement initiatives.

Building this understanding is especially important in improving existing maintenance processes. Accumulated project knowledge and expertise might be lost if a prescriptive software maintenance process is put in place without regard to existing processes and practices. In addition, a prescriptive process that is not adequately tailored for its environment may encounter organizational and personnel roadblocks. If the prescriptive process is seen as a distillation of 'best practices', then the characterization and evaluation of current processes is a necessary first step. Such a descriptive model baseline is also necessary to evolve an effective process (Curtis, Kellner and Over, 1992).

Establishing a measurement program integrated into the maintenance process is likely to help any organization achieve an in-depth understanding of its specific maintenance issues and thereby lay a solid foundation for maintenance process improvement (Rombach, Ulery and Valett, 1992). However, defining and enacting a measurement program takes time. A short term, quickly operational substitute is needed in order to obtain a first quick insight, at low cost, into the issues to be addressed. Furthermore, defining efficient and useful measurement procedures first requires a characterization of the maintenance environment in which measurement takes place, such as organization structures, processes, issues and risks (Basili and Rombach, 1988).

Part of this characterization is the identification and evaluation of issues that must be addressed in order to improve the quality and productivity of maintenance projects. Because of the complexity of the phenomena studied, this is a difficult task for the maintenance organization (Hariza *et al.*, 1992). Each project may encounter specific difficulties and situations that are not necessarily alike across all the organization's maintenance projects. This may be due in part to variations in design and code decay, application domain, size, change frequency, and/or schedule and budget constraints. As a consequence, a representative sample of projects must first be analysed as separate entities even if, later on, commonalities across projects may require similar solutions for improvement. In addition, much of the data collected for characterization will be of a qualitative nature, unless historical quantitative maintenance data are available. Given these conditions, it is only natural to make use of qualitative research methods, which aim to provide understanding and explanations by studying selected issues based on information of a qualitative nature (Patton, 1990).

In this paper, we propose a qualitative, inductive method for characterizing and evaluating software maintenance processes, thereby identifying their specific problems and needs. This method encompasses a set of procedures which attempt to determine causal links between maintenance problems and flaws in the maintenance organization and process. This allows for a set of concrete steps to be taken for maintenance quality and productivity improvement, based on a tangible understanding of the relevant maintenance issues in a particular maintenance environment. Moreover, this understanding provides a solid basis on which to define relevant software maintenance models and measures. A case study of the application of this method, called Q-MOPP, is presented to further illustrate its feasibility and benefits. Currently Q-MOPP focuses on the project level, although future work is planned to enable project-level results to be scaled up. This is a

similar approach to that taken in the SEI's CMM, for example, where improvement issues are first addressed at the project level, then at the organization level.

Section 2 briefly presents the background techniques we use. Section 3 presents our method, the modelling techniques it uses, and its data acquisition procedures. It is then further detailed and illustrated in Section 4 through a selected case study. Section 5 summarizes our experience and lessons learned with using this method on several case studies and Section 6 provides the main conclusions of the paper.

2. BACKGROUND

2.1. Descriptive process modelling

In the process modelling literature, process models are often classified as prescriptive, descriptive or proscriptive (Curtis, Kellner and Over, 1992). Much of the research in process modelling has been from the prescriptive modelling perspective, usually presenting modelling formalisms, e.g., STATEMATE, or software engineering environments that allow enactment of process programs, e.g., Marvel. However, it is not always clear how the prescriptive model was arrived at, or how it was judged that a prescriptive model was appropriate for a particular situation. A number of process improvement studies do exist that synthesize a recommended development process from existing processes (Bandinelli *et al.*, 1995). A necessary activity in such studies is the description and understanding of the processes in place as well as related aspects such as organization and product. This activity is referred to as descriptive process modelling. Recently there has been research focusing on descriptive modelling, and more precisely the systematic and unbiased elicitation of process models (Madhavji *et al.*, 1994).

The construction of descriptive models is especially important for evolutionary improvement, by providing baselines, and more importantly, supplying a rich data source for in-depth understanding of the process in place. Previous research, whether in software maintenance or the software process, that has focused on understanding, has had a strong descriptive component to it, and has employed qualitative research methods (Bendifallah and Scacchi, 1987; Swanson and Beath, 1988; Curtis, Krasner and Iscoe, 1988; Dart, Christie and Brown, 1993). Developing a good descriptive model should combine both qualitative and quantitative research methods, since both types of data must be collected and analysed.

2.2. Qualitative evaluation

Qualitative research consists of qualitative data collection and qualitative analysis. Qualitative data are data in the form of words and pictures, not numbers (Gilgun, 1992). There are three major modes of collecting qualitative data: participant observation, interviewing and studying materials prepared by others (Wolcott, 1994). In participant observation, the researcher is not actively participating, but is an observer whose presence is apparent to those being studied. The researcher takes field notes of what is observed. Interviewing can range from unstructured to structured, depending on the instrumentation

used. Structured interviews make use of questionnaires, while unstructured interviews do not, and using interview guides falls somewhere in between. Manuals, handbooks and code are some of the material from which data may also be collected.

A good overview of qualitative analysis methods is given by Miles and Huberman (1994) which presents qualitative analysis as three concurrent flows of activity: data reduction, data display, and conclusion drawing and verification. Abstracting the data, displaying the data in pictorial form and constructing hypotheses based on the data are all descriptive and exploratory analysis activities. Analysis may also be evaluatory when its purpose is to see how a process might be improved, or why it is not producing the expected results (Wolcott, 1994).

Much of the data used to build process and organizational models are of a qualitative nature, such as those collected from interviews with process participants, or examination of development or maintenance artefacts, e.g., error report forms, process documents, project deliverables. It is also the case that qualitative data must be collected when no measurement plan exists, or when it is not clear what should be measured. In these circumstances it is only natural to employ qualitative methods for evaluation. Such qualitative evaluation is routinely used in fields such as education or health care, where the impact or effectiveness of interventions on individuals must be judged. The aim of such evaluations is to provide practical suggestions for improvement and change (Patton, 1990).

3. MAINTENANCE EVALUATION METHOD

3.1. Overview

The following subsections present the details of the evaluation method we propose, Q-MOPP. Subsection 3.2 first presents an overview of the main components of the method. Subsection 3.3 describes its steps and subsection 3.4 their supporting techniques. Subsection 3.5 provides guidelines for the collection of data within each step.

3.2. Main components

Q-MOPP has two phases: descriptive modelling and analysis. Conceptually, we include three components for descriptive modelling: organization, process and product. Explicit modelling techniques are used to model the organization and process, while the product is represented implicitly as process artefacts or organizational information flows. In addition, the content and structure of products must be described explicitly to a suitable level of detail, although they are not presented in our case study for the sake of brevity. The result of descriptive modelling is a comprehensive description of what is actually happening during the creation of a new maintenance release of the system. The analysis phase uses this comprehensive descriptive model to identify areas in need of improvement and to provide improvement suggestions. Analysis is focused on the software defects that were generated during the maintenance process, because we consider these defects to be related to the maintenance process itself. The complete application of the maintenance

evaluation method will provide specific information regarding its problems and improvement needs, an understanding of the measures that are relevant to the maintenance organization, and insights into what could be adequate data collection procedures. Table 1 presents the conceptual model of our method.

3.3. Steps of the method

3.3.1. Six-step method

Based on the conceptual model in Table 1, we break down the maintenance evaluation process into six steps. The steps are organized so that each step builds on the previous ones, and so some steps collect a combination of organization, process or product data. However, the application of these steps is likely to require several iterations and a stepwise refinement of previous steps' outputs at the completion of each step.

3.3.2. Step 1: organizational modelling

Identify the organizational entities with which the maintenance team interacts and the organizational structure in which maintainers operate. In this step the distinct teams, working groups and their roles in the change process are identified. Information flows between actors are also determined.

3.3.3. Step 2: Identify phases

Identify the phases involved in the creation of a new system release. The notion of phase has, in our context, a very different definition than that of activities (Step 3). Phases produce one or several intermediate or final release products that are reviewed according to quality assurance procedures, when they exist, and are officially approved. In addition, the phases of a release are ordered in time, although they may be somewhat

Table 1. Conceptual model of Q-MOPP

Phase	Focus	Activities
Descriptive modelling	Organization	Identify organizational entities, structures. Identify actors in organization. Determine information flows between actors.
	Process (release generation)	Identify phases in creation of new system release. Identify activities in each phase. Map actors to phases and activities.
	Product	Identify software artefacts produced and consumed by each phase. Map software artefacts to informaton flow between actors.
Analysis	Defect	Select representative past release(s) to be analysed. Analyse maintenance problems. Relate problems to organization and process flaws.

overlapping, and are clearly separated by milestones. Software artefacts produced and consumed by each phase must be identified. Actors responsible for producing and validating the output artefacts of each phase must be identified and located in the organizational structure defined in Step 1.

3.3.4. Step 3: Identify activities

Identify the generic activities involved in each phase, i.e., decompose the phases identified in Step 2 to a lower level of granularity. Identify, for each low-level activity, its inputs and outputs and the actors responsible for them. Activities are tasks which cannot be a priori ordered within a phase and do not produce deliverables going through a formal approval process, although they can be reviewed (e.g., peer reviews). Phases contain activities but activities may belong to several phases, e.g., coding may take place during requirements analysis (e.g., prototyping) as well as during implementation.

3.3.5. Step 4: Select release(s)

Select one or several representative past releases for analysis (preferably from different systems to allow for better generalization of the results). Acquire available documentation about each selected release.

3.3.6. Step 5: Analyze defects

Based in part on release documents and error report forms, analyse the defects that occurred while performing the software changes in the selected releases in order to produce a causal analysis document. The knowledge and understanding acquired through Steps 1-3 are necessary in order to understand, interpret and formalize the information in the causal analysis document.

3.3.7. Step 6: Analyze defect distributions

Establish the frequency and consequences of problems due to flaws in the organizational structure and the maintenance process by further analysing the information gathered in Step 5.

3.3.8. A learning paradigm

The process described above focuses on a set of releases for a given maintained system. In order to draw conclusions not only at the system level but also at an organizational level, this analysis needs to be repeated on a number of systems which are representative of the organization's maintenance activities. It should be noted that different conclusions may be drawn for different systems since they may vary in age, application domain, documentation level and quality, and so on.

Our maintenance evaluation process is essentially an instantiation of the generic qualitative analysis process defined in Shelly and Sibert (1992). Figure 1 illustrates at a high level our maintenance-specific qualitative analysis process. It is a combination of both

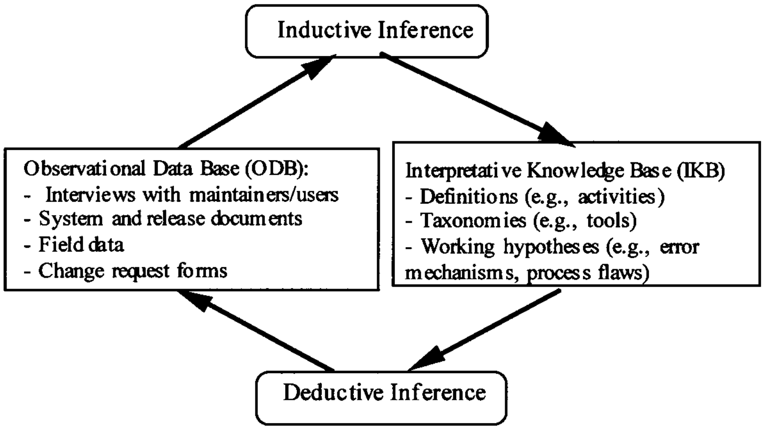

Figure 1. The paradigm for our qualitative analysis for software maintenance has two inference processes utilizing two stores, one of data and one of models (Briand et al.*, 1994)*

inductive and deductive inferences. The collected information, such as field notes or interviews, comprise the *observational database* (ODB). Inductive inferences are made from the collected information, resulting in models, which are then stored in the *interpretative knowledge base* (IKB). Deductive inferences occur when, based on our IKB, we derive expectations about the real world. An example of such an expectation might be that all errors can be exhaustively classified according to defined taxonomies. When comparing these expectations with new field information feeding the ODB, we can experimentally validate and refine the knowledge in the IKB (e.g., a defect taxonomy). Then the data collection process is refined in order to resolve ambiguities and answer new questions, which leads to refined and revised inductive inferences. The process continues in an iterative fashion. This iterative pattern not only applies to the overall evaluation process, but also to several of the individual steps. For example, in our case study, performing Step 1 revealed additional issues to be addressed in building an organizational model, and led to the selection and use of a more sophisticated modelling approach.

3.4. Techniques

3.4.1. Organizational modelling

In the preceding description of our maintenance evaluation method we did not mention specific techniques, such as modelling approaches or tools, to be used to support modelling and analysis activities. In this subsection we present the techniques, including taxonomies, we use in our application of the method.

The first step of the evaluation method requires a technique that helps us capture and build organization models. After examining the process literature, Yu's actor–dependency (A–D) modelling technique (Yu and Mylopoulos, 1994) was selected. This technique

specifies well, is based on a clearly defined framework, and can model how different types of people and roles interact. A–D models are based on process participants and the many types of relationships between them, including information flow relationships. A–D models also provide a mechanism for representing members of the organization in a variety of ways that we believe is based on a clear and convenient paradigm. In Briand *et al.* (1995) we provided a detailed list of enhancements we proposed to this technique, based on our experience.

A basic actor–dependency model represents an organizational structure as a network of dependencies among organizational entities, or actors. We have also used the agent–role–position (ARP) model, an extension of the A–D model, to further decompose actors themselves.

A node in an A–D network represents an organizational actor, and a link indicates a dependency between two actors. Examples of actors are: someone who inspects units, a project manager or the person who gives authorization for final shipment. Documents to be produced, goals to be achieved and tasks to be performed are examples of dependencies between actors. When an actor, A1, depends on A2, through a dependency D1, it means that A1 cannot achieve, or cannot efficiently achieve, its goals if A2 is not able or willing to fulfil its commitment to D1. The A–D model provides four types of dependencies between actors:

(1) In a *goal dependency*, an actor (the depender) depends on another actor (the dependee) to achieve a certain goal or state, or fulfil a certain condition (the dependum). The depender does not specify how the dependee should do this. A fully built configuration, a completed quality assessment or 90% test coverage of a software component might be examples of goal dependencies if no specific procedures are provided to the dependee(s).
(2) In a *task dependency*, the depender relies on the dependee to perform some task. This is very similar to a goal dependency, except that the depender specifies how the task is to be performed by the dependee, without making the goal to be achieved by the task explicit. Unit inspections are examples of task dependencies if specific standard procedures are to be followed.
(3) In a *resource dependency*, the depender relies on the dependee for the availability of an entity (physical or informational). Software artefacts (e.g., designs, source code, binary code), software tools, documents and any kind of computational resources are examples of resource dependencies.
(4) A *soft-goal dependency* is similar to a goal dependency, except that the goal to be achieved is not sharply defined, but requires clarification between depender and dependee. The criteria used to judge whether or not the goal has been achieved is uncertain. Soft goals are used to capture informal concepts which cannot be expressed as precisely defined conditions, as are goal dependencies. High product quality, user friendliness and user satisfaction are common examples of soft goals because, in most environments, they are not precisely defined.

Dependencies are usually modelled from the depender's perspective since the depender

is a priori in a better position to state their needs. However, if time allows, the dependee's perspective on the same dependencies may also be captured and compared for consistency.

Figure 2 shows a simple, fictitious, example of an A–D model. A manager oversees a tester and a developer. The manager depends on the tester to test. This is a task dependency because there is a defined set of procedures that the tester must follow. In contrast, the manager also depends on the developer to develop, but the developer has complete freedom to follow whatever process she or he wishes, so this is expressed as a goal dependency. Both the tester and the developer depend on the manager for positive evaluations, where there are specific criteria to define 'positive', thus these are goal dependencies. The tester depends on the developer to provide the code to be tested (a resource), while the developer depends on the tester to test the code well (good coverage). Assuming that there are no defined criteria for 'good' coverage, this is a soft-goal dependency.

In the previous paragraphs, what we referred to as an actor is in fact a composite notion that can be refined in several ways to provide different views of the organization. *Agents*, *roles* and *positions* are three possible specializations of the notion of actor which are related as follows:

(5) an agent occupies one or more positions,
(6) an agent plays one or more roles, and
(7) a position can cover different roles in different contexts.

A more detailed description of the A–D model may be found in Yu and Mylopoulos (1994). It is important to note that, in practice and in the context of Q-MOPP, some types of dependencies may not appear in a given organization. For example, if the maintenance process is largely unspecified or undocumented, then there are not likely to be very many task dependencies between agents.

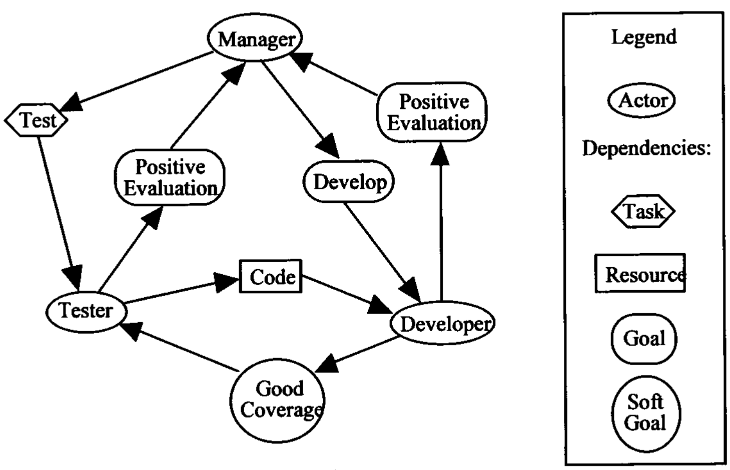

Figure 2. This simple example of an A–D model shows the actors and their dependencies

3.4.2. Process modelling

We did not use any particular modelling formalism to construct a descriptive model of the maintenance (release generation) process. We did, however, develop taxonomies to aid us in organizing process information. As part of process modelling we distinguished phases and activities in the process. In our experience this was adequate, but our method does not preclude the use of process modelling formalisms. The case study in Section 4 illustrates how we used the taxonomies and the data collected to model the process.

The taxonomy of generic maintenance activities is shown in Figure 3. Figure 4 shows a taxonomy to help characterize the maintenance tools and techniques under study. The taxonomy shows only the first level of abstraction, so that it can be specialized for a particular maintenance environment.

The process model based on these taxonomies consists of phases in the process, with the associated inputs to and outputs from, and activities in the phase. In our case study in Section 4, such a process model is represented in tabular form.

3.4.3. Product modelling

By 'product', we mean all artefacts created during the maintenance process, such as code, test plans and other release documentation. Product modelling is not, in the current

Acronym	Activity
DET	Determination of the need for a change
SUB	Submission of change request
UND	Understanding requirements of changes: localization, change design prototype
IA	Impact analysis
CBA	Cost/benefit analysis
AR	Approval/rejection/priority, assignment of change request
SC	Scheduling/planning of task
CD	Change design
CC	Code changes
UT	Unit testing of modified parts, i.e., has the change been implemented?
IC	Unit Inspection, Certification, i.e., has the change been implemented properly and according to standards?
IT	Integration testing, i.e., does the changed part interface correctly with the reconfigured system?
RT	Regression testing, i.e., does the change have any unwanted side effects?
AT	Acceptance testing, i.e., does the new release fulfill the system requirements?
USD	Update system and user documentation
SA	Checking conformance to standards; quality assurance procedures
IS	Installation
PIR	Post-installation review of changes
EDU	Education/training regarding the application domain/system

Figure 3. Our taxonomy of generic maintenance activities is adapted from Bennett et al. *(1991) and Harjani and Queille (1992)*

Maintenance tools:

- impact analysis and planning too
- tools for automated extraction and representation of control and data flows
- debugger
- generator of cross-references
- regression testing environment (data generation, execution, and analysis of results)
- information system linking documentation and code

Maintenance techniques:

- rigorous impact analysis, planning, and scheduling procedures
- systematic and disciplined update procedures for user and system documentation
- user communication channels and procedures

Figure 4. This first–level abstraction of taxonomies is adapted from Bennett et al. *(1991)*

state of our method, defined in a stepwise manner as are the organization and process modelling tasks. It basically consists of identifying the life cycle products' structure, content and interdependencies. In addition, for each product, its various states and the transitions between them should be identified and associated with the completion of process activities or phases. Using the organization model, responsibilities and roles regarding the product should be specified. A generic taxonomy for non-code artefacts is shown in Figure 5. In addition to the artefacts considered in the generic taxonomy, there might be artefacts that are specific to a process, such as materials prepared for inspection meetings.

3.4.4. Causal analysis

The analysis phase of our maintenance evaluation method begins with product defects. More specifically, we examine reported defects in a system release, paying particular attention to those defects the maintainers themselves considered to be significant in some way. Using the descriptive models already constructed, the defects are mapped to human errors, which are in turn mapped to flaws in the overall maintenance process (organization, release generation process, product). Figure 6 presents the conceptual model for the analysis: flaws in the overall maintenance process lead to human errors, which in turn lead to product defects.

To aid us in this mapping, we defined taxonomies for categorizing errors and flaws. These taxonomies are based on widely accepted definitions as well as our own observations. Figure 7 shows a taxonomy of human errors. Determining the origin and cause of errors helps to determine their possible causal relationships to overall maintenance process flaws.

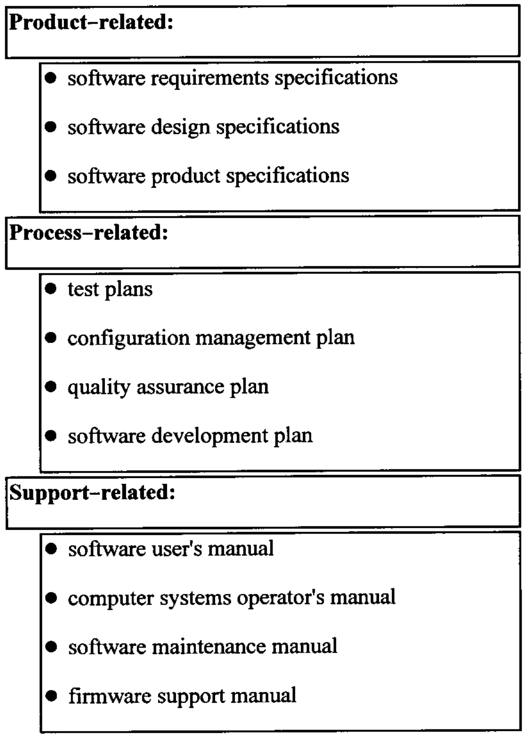

Figure 5. This generic taxonomy of maintenance documentation is adapted from Bennett et al. *(1991)*

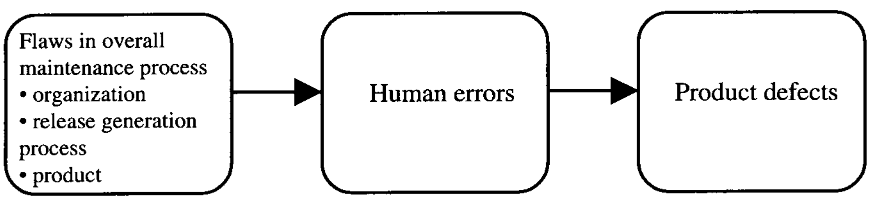

Figure 6. Product defects are traced to process flaws via human errors

This taxonomy is shown in Figure 8. In Subsection 4.3, we present details of how the causal analysis was performed for the case study, illustrating how this part of Q-MOPP can be extended for particular applications of this method.

3.5. Data acquisition procedures

3.5.1. Organizational and product data

Below, we describe in more detail the collection of various types of data needed to support the execution of Q-MOPP. First, we describe the procedure adopted to collect

Error origin: when did the misunderstanding occur?

- Change requirements analysis
- Change localization analysis
- Change design analysis
- Coding

Error domain: what caused it?

- Lack of application domain knowledge: *operational constraints (user interface, performance), mathematical model*
- Lack of system design or implementation knowledge: *data structure or process dependencies, performance or memory constraints, module interface inconsistency*
- Ambiguous or incomplete requirements
- Language misunderstanding <semantic, syntax>
- Schedule pressure
- Existing uncovered fault
- Oversight

Figure 7. The taxonomy of human errors has two parts

organizational and process information as well as a brief discussion on product information. We use the techniques described in the previous section to structure and model that information. Second, the procedure to collect information regarding release changes is presented.

Any modelling effort requires that a great deal of information be collected from the environment being modelled. Building an A–D model requires collecting information about many people in the environment, the details of their jobs and assignments, whom they depend on to complete their tasks and reach their goals, etc. Product information also has to be captured, to the extent that it describes the content of information dependencies between actors. The process we followed, with modifications motivated by our experience, is briefly presented below:

1.1 First, we determine the official, (usually) hierarchical structure of the organization. Normally this information can be found in official organization charts. This gives us the set of positions and the basic reporting hierarchy.

1.2 We determine the roles covered by the positions by interviewing the people in each position, and then, to check for consistency, their supervisors and subordinates. Process descriptions, if available, often contain some of this information. However,

Organizational flaws

- communication: interface problems, information flow "bottlenecks" in the communication between the maintainers and the
  - users
  - management hierarchy
  - quality assurance (QA) team
  - configuration management team
- roles:
  - prerogatives and responsibilities are not fully defined or explicit
  - incompatible responsibilities, e.g., development and QA
- process conformance: no effective structure for enforcing standards and processes

Maintenance methodological flaws

- Inadequate change selection and priority assignment process
- Inaccurate methodology for planning of effort, schedule, personnel
- Inaccurate methodology for impact analysis
- Incomplete, ambiguous protocols for transfer, preservation and maintenance of system knowledge
- Incomplete, ambiguous definitions of change requirements
- Lack of rigor in configuration (versions, variations) management and control
- Undefined / unclear regression testing success criteria

Resource shortages

- Lack of financial resources allocated, e.g., necessary for preventive maintenance, unexpected problems unforeseen during impact analysis.
- Lack of tools providing technical support (see previous tool taxonomy)
- Lack of tools providing management support (i.e., impact analysis, planning)

Low quality product(s)

- Loosely defined system requirements
- Poor quality design, code of maintained system
- Poor quality system documentation
- Poor quality user documentation

Personnel-related issues

- Lack of experience and/or training with respect to the application domain
- Lack of experience and/or training with respect to the system requirements (hardware, performance) and design
- Lack of experience and/or training with respect to the users' operational needs and constraints

Figure 8. The taxonomy of maintenance flaws includes resource shortages, low quality products and personnel-related issues

when using process descriptions, the modeller must check carefully for process conformance.

1.3 In this step, we focus on the goal, resource and task dependencies that exist along the vertical links in the reporting hierarchy. To do this, we interview members of different departments or teams, as well as the supervisors of those teams. Also, direct observation of supervisors, called ‘shadowing’, can be useful in determining

exactly what is requested of, and provided by supervisors for their subordinates. If any resource dependencies are identified in which the dependum is a product, it must be characterized using the document taxonomy in Section 3.4.3.

1.4 Next we focus on dependencies between members of the same team. Direct observation (through shadowing or observation of meetings) is also useful here. Interviews and process documents can also be used to identify dependencies. Again, any products involved must be characterized.

1.5 Finally, we determine the dependencies between different teams. These are often harder to identify, as they are not always explicit. Direct observation is especially important here, as often actors do not recognize their own subtle dependencies on other teams. It is also very important in this step to carefully check for mutual dependencies between actors who work in different parts of the management hierarchy. Any products that flow between different teams must also be characterized.

3.5.2. Product and process data

Product modelling is even more interrelated with process modelling than with organization modelling. Information about products is embedded in the process model in the descriptions of phase inputs and outputs. Product information is collected by examining release documentation (e.g., user's guide, the system description and material prepared for release meetings) in order to obtain information regarding, for example, the size of the products, their structure and their interdependencies. As above, in the description of organization data collection, here we describe the gathering of product information as it is related to the building of the process model.

Below are the steps we followed to identify the process phases involved in the creation of a new system release (Step 2):

2.6 Identify the phases as defined in the environment studied. At this stage, it is important *not* to map an *a priori* external/generic maintenance process model and vocabulary.

2.7 Each product (e.g., document, source code) which is input or output of each phase has to be determined and its content carefully described (see document taxonomy in Section 3.4.3).

2.8 The personnel in charge of producing and validating the output artefacts of each phase have to be identified and located in the organizational structure defined in Step 1.

These are the steps we followed to identify the generic activities involved in each phase (Step 3):

3.1 Select from the literature (Chapin, 1987; Bennett *et al.*, 1991; Harjani and Queille, 1992) or define and tailor a taxonomy of generic activities based on widely accepted definitions and used in the maintenance process. As a guideline, such a taxonomy was proposed in Section 3.4.2.

3.2 Map these activities into each phase by reading the technical documents produced and interviewing the technical project leaders and maintainers about their real work habits. If possible, collect effort data for each activity so that the importance of each activity in each phase can be assessed somewhat quantitatively.

3.5.3. Change data for causal analysis

Ideally, it is most useful to analyse problems as they are occurring and thereby better understand process and organization flaws. However, because of time constraints, it is often more practical to analyse past releases. We present below a set of guidelines for selecting them:

- Recent releases are preferable since maintenance processes and organizational structure might have changed and this would make analyses based on old releases somewhat irrelevant.
- Some releases may contain more complete documentation than others. Documentation has a very important role in detecting problems and cross checking the information provided by the maintainers.
- The technical leader(s) of a release may have left the company whereas another release's technical leader may still be contacted. This is a crucial element since, as we will see, the change analysis process will involve project technical leader(s) and, depending on their level of control and knowledge, possibly the maintainers themselves.

Because our particular evaluation model was based on the analysis of maintenance changes, we had to collect information regarding each maintenance change. An outline of the information that we collected for each software change is provided in Figure 9. This information was acquired by interviewing the maintainers and technical leaders and by reading the related documentation: release intermediate and final deliverables, error report forms from system and acceptance test. A questionnaire was also used to gather additional information regarding changes that were part of the release studied. The questionnaire used the taxonomies presented in Figure 8.

4. CASE STUDY

4.1. Objective

The objective of this section is two-fold: to illustrate the application of the Q-MOPP method through a case study and to demonstrate its applicability. We first describe the project selected in this case study and continue by describing the main aspects of the models we built, the analyses we performed and the results we obtained.

4.2. Maintenance project

Our method was applied to three projects in the Flight Dynamics Division (FDD) of NASA Goddard Space Flight Center (Condon *et al.*, 1995), one of which is the subject

1 Description of the change

1.1 Localization
- subsystem(s) affected
- module(s) affected
- inputs/outputs affected

1.2 Size
- LOCs deleted, changed, added
- Modules examined, deleted, changed, added

1.3 Type of change
- Preventive changes: improvement of clarity, maintainability or documentation.
- Enhancement changes: add new functions, optimization of space/time/accuracy
- Adaptive changes: adapt system to change of hardware and/or platform
- Corrective changes: corrections of development errors.

2 Description of the change process

2.1 effort, elapsed time

2.2 maintainer's expertise and experience
- How long has the person been working on the system?
- How long has the person been working in this application domain?

2.3 Did the change generate a change in any document? Which document(s)?

3 Description of the problem

3.1 Were some errors committed?
- Description of the errors (see taxonomies in Figure 7)
- Perceived cause of the errors: maintenance process flaw(s) (see Figure 8)

3.2 Difficulty
- What made the change difficult?
- What was the most difficult activity associated with the change?

3.3 How much effort was wasted (if any) as a result of maintenance process flaws?

3.4 What could have been done to avoid some of the difficulty or errors (if any)?

Figure 9. The data collected covers three main areas

of the case study presented in this paper. Our case study takes place in the framework of the NASA Software Engineering Laboratory (NASA-SEL), an organization aimed at improving FDD software development processes based on measurement and empirical analysis.

Measurement of maintenance in the SEL began in late 1987; there was little documentation of maintenance procedures and lack of explicitly documented experience in the maintenance environment at that time (Rombach and Ulery 1989; Rombach, Ulery and Valett, 1992). Our study of maintenance processes, focusing on qualitative analysis methods, was begun in late 1993. Partial research results were reported by Briand *et al.* (1994), Briand *et al.* (1995) and Basili *et al.* (1996).

GTDS, a 28 year old, 250 KLOC, FORTRAN orbit determination system, was selected for our case study. GTDS was selected mainly because it was representative of the type of systems developed and maintained in the environment under study (i.e., application domain, programming language) and was furthermore a critical organizational asset. GTDS is public domain software, with a large group of users all over the world. Usually one or two releases are produced each year, in addition to mission-specific versions not immediately subject to configuration management but which are later integrated into a new version by going through the standard release process. Compared with other projects in the maintenance organization, turnover in the GTDS maintenance team was low.

In this case study, the GTDS release selected for causal analysis was quite recent, most of the documentation identified in Step 2 was available, and most importantly, the technical leader of the release was available for additional insights and information.

4.3. Descriptive models

4.3.1. The organization model

The descriptive models presented are the results of carrying out Steps 1 to 3 of our method. The organization model is represented as an A–D model. The process model is represented as a sequence of phases along with the activities in each phase. These two models present complementary information. Products are present in these models in the form of resources in the A–D model, and of inputs and outputs of phases in the process.

The organizational model in Figure 10 is very complex despite important simplifications (e.g., agents and roles are not included). This shows how intricate the network of dependencies in a large software maintenance organization can be.

The model is by necessity incomplete. We have focused on those positions and activities that contribute to the maintenance process only. So there are many other actors in the NASA-FDD organization which do not appear in the A–D graph. As well, we have aggregated some of the positions where appropriate. For example, maintenance management actor includes a large number of separate actors, but for the purposes of our analysis, they can be treated as an aggregate. Below are listed the positions shown in the figure, and a short explanation of their specific roles:

- *Testers* present acceptance test plans, perform acceptance test and provide change requests to the maintainers when necessary.

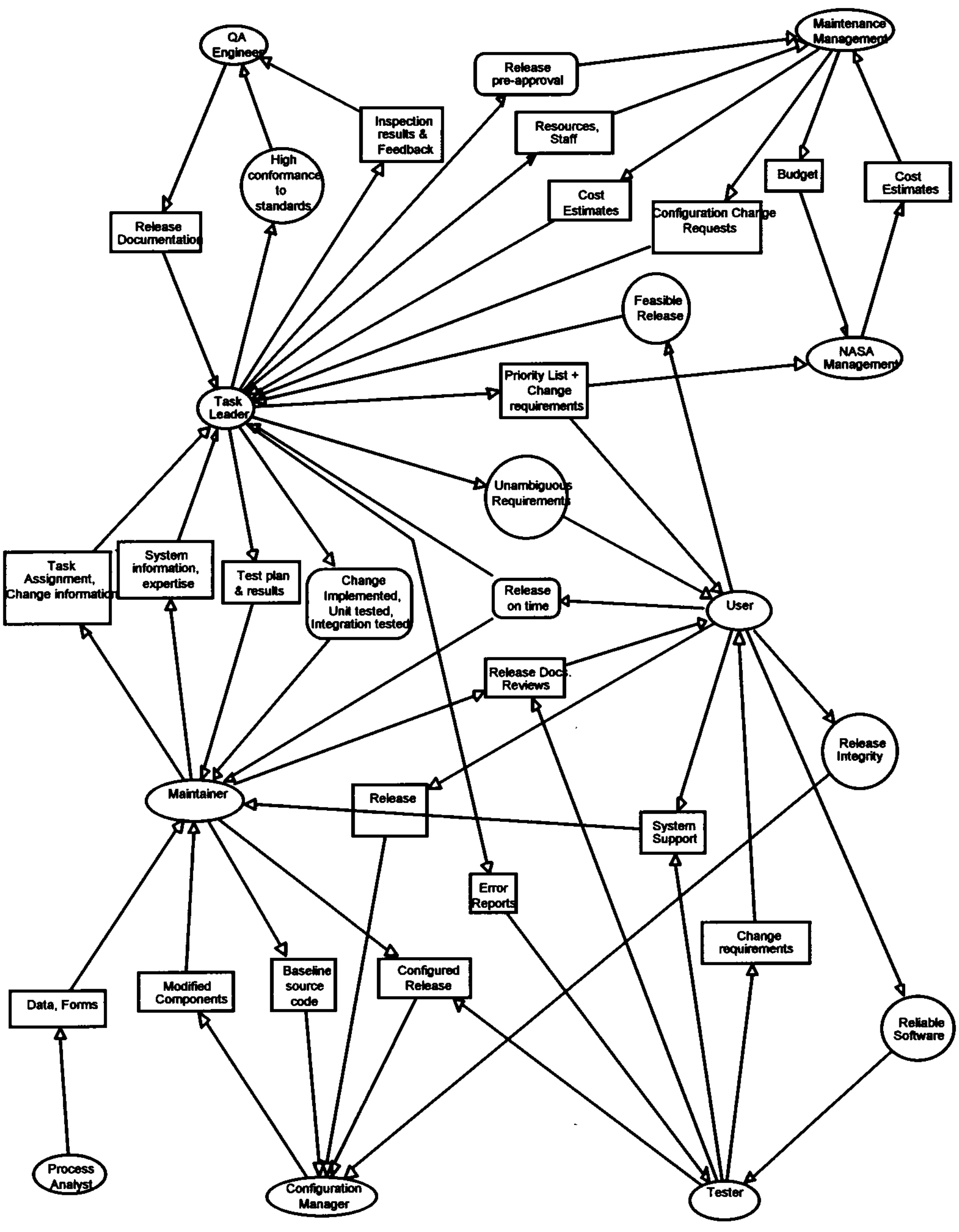

Figure 10. The complexity of the A–D model is not exceptional

- *Users* suggest, control and approve performed changes.
- *QA engineer* controls maintainers' work (e.g., conformance to standards), attends release meetings and audits delivery packages.
- *Configuration manager* integrates updates into the system, co-ordinates the production and release of versions of the system and provides tracking of change requests.
- *Maintenance management* grants preliminary approvals of maintenance change requests and release definitions.
- *Maintainers* analyse changes, make recommendations, perform changes, perform unit and change validation testing after linking the modified units to the existing system, perform validation and regression testing after the system is recompiled by the configuration manager.
- *Process analyst* collects and analyses data from all projects and packages data to be reused.
- *NASA management* is officially responsible for selecting software changes, gives official authorizations and provides the budget.

The resulting organization model was validated through use, within the context of the maintenance evaluation method. The modelling of the maintenance process, the release documents and the causal analysis of maintenance problems allowed us to check the model for consistency and completeness.

4.3.2. The process model

The process shown in Figure 11 represents our partial understanding of the working process for a release of GTDS and the mapping into standard generic activities (using the taxonomy in Figure 3). This combines the information gained from Steps 2 and 3 of the assessment process. Activity acronyms are used as defined in Figure 3.

In this case, each phase milestone in a release is represented by the discussion, approval and distribution of a specific release document (which are defined in the taxonomy shown in Figure 5). The resulting process model was validated through consulting with the GTDS task leader and another maintainer. Validation questions include:

- Are all the people in the process model a part of the organization model?
- Do the documents and artefacts included in the process model match those of the information flow of the organization model?
- Is the mapping between activities and phases complete, i.e., an exhaustive set of activities, a complete mapping?
- Are a priori relevant types of activities (e.g., defined in Figure 3) missing from the process model?

4.4. Change causal analysis

4.4.1. Step 5 subtasks

Step 5 involved the identification and causal analysis of the problems observed during the maintenance and acceptance test of the release studied. These problems were linked

Phase 1. Change analysis	
Input:	change requests from software owner and priority list
Output:	Release Content Review (RCR) document which contains change design analysis, prototyping, and cost/benefit analysis that may result in changes in the priority list that will be discussed with the software owner/user.
Activities:	UND, IA, CBA, CD, some CC, UT and IT for prototyping

Phase 2. RCR meeting	
Input:	draft of Release Content Review document proposed by maintainers is discussed, i.e., change priority, content of release.
Output:	Updated Release Content Review document
Activities:	AR, SA (QA engineers are reviewing the release documents and attending the meeting)

Phase 3. Solution analysis	
Input:	updated Release Content Review document
Output:	devise technical solutions based on prototyping analysis they performed in Step 1, Release Design Review (RDR) document.
Activities:	SC, CD, CC, UT (prototyping), (preparation of test strategy for) IT (based mainly on functional testing: equivalence partitioning)

Phase 4. RDR meeting	
Input:	RDR documentation
Output:	approved (and possibly modified) RDR documentation
Activities:	review and discuss CC, UT, (plan for) IT, SA

Phase 5. Change implementation and test	
Input:	RDR and prototype solutions (phases 1, 3)
Output:	changes are completed and unit tested; change validation test is performed with new reconfigured system (integration test); formal inspections are performed (when quality of code and design allows it) on new or extensively modified components; some (usually superficial) regression testing is performed on the new system to insure minimal quality before AT; a report with the purpose of demonstrating that the system is ready for AT is produced: Acceptance Test Readiness Review document (ATRR)
Activities:	CC, UT, IC, IT, RT, USD, SA

Figure 11. The process model consists of seven phases

Phase 6. ATRR meeting	
Input:	Acceptance Test Readiness Review document
Output:	The changes are discussed and validated and the testing strategy used is discussed. The acceptance test team presents its acceptance testing plan.
Activities:	review the current output of IT, SA

Phase 7. Acceptance test	
Input:	the new GTDS release and all release documentation
Outputs:	A list of Software Change Requests (SCRs) is provided to the maintainers. These changes correspond to inconsistencies between the new reconfigured release and the general system requirements.
Activities:	RT, AT

Figure 11. Continued

back to a precise set of issues belonging to taxonomies presented in Figures 6 and 7. Figure 12 summarizes Step 5 as instantiated for this case study. This step required extensive collaboration from the GTDS maintenance task leader, as well as examination of the documents generated during the release process. Changes that generated error correction requests from the acceptance testing team were analysed in particular detail.

Step 5 can be broken down into a sequence of subtasks. These subtasks should be performed for each software change examined:

5.1 Determine the difficulty or error-proneness of the change.
5.2 Determine whether and how the change difficulty could have been alleviated, or the error(s) resulting from the change avoided.
5.3 Estimate the size of the change (e.g., number of components, lines of code changed, added, removed).
5.4 Assess discrepancies between initial and intermediate planning and actual effort and time.

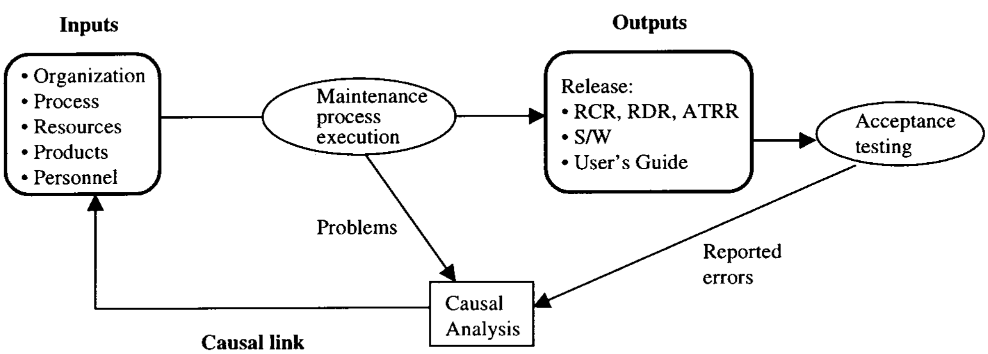

Figure 12. The change analysis process provides for feedback and iteration (Briand et al., *1994)*

5.5 Determine the human flaw(s) (if any) that originated the error(s) or increased the difficulty related to the change (using the taxonomy shown in Figure 7).
5.6 Determine the maintenance process flaws that led to the identified human errors (if any), using the taxonomy of maintenance process flaws proposed in Figure 8.
5.7 Try to quantify the wasted effort and/or delay generated by the maintenance process flaws (if any).

In order to illustrate Step 5, we provide below an example of change analysis for *one* of the changes in the selected release (change 642). Implementation of this change resulted in 11 errors that were found by the acceptance test team, eight of which had to be corrected before final delivery could be made. In addition, a substantial amount of rework was necessary. Typically, changes do not generate so many subsequent errors, but the flaws that were present in this change are representative of maintenance problems in GTDS. In the following paragraphs as in Briand *et al.* (1994), we discuss only *two* of the errors generated by the change studied (errors A1044 and A1062).

4.4.2. Change 642

Description Initially, users requested an enhancement to existing GTDS capabilities. The enhancement involved vector computations performed over a given time span. This enhancement was considered quite significant by the maintainers, but users failed to supply adequate requirements and did not attend the release content review (RCR) meeting. Users did not report their dissatisfaction with the design until the acceptance test readiness review (ATRR) meeting time, at which time requirements were rewritten and maintainers had to perform rework on their implementation. This change took a total of three months to implement, of which at least one month was attributed to rework.

Maintenance process flaw(s) Organizational. A lack of clear definitions of the prerogatives/duties of users with respect to release document reviews and meetings (roles), and a lack of enforcement of the release procedure (process conformance).

Methodological. Incomplete, ambiguous definitions of change requirements.

Errors caused by change 642 The implementation of the change itself resulted in an error (A1044) found at the acceptance test phase. When the correction to A1044 was tested, an error (A1062) was found that could be traced back to both 642 and A1044.

Error A1044 Description. Vector computations at the endpoints of the time span were not handled correctly. But in the requirements it was not clear whether the endpoints should be considered when implementing the solution.

Error origin. Change requirement analysis.

Error domain. Ambiguous and incomplete requirements.

Maintenance process flaw(s).

- Organizational: communication between users and maintainers, due in part to a lack of defined standards for writing change requirements;
- Methodological: incomplete, ambiguous definitions of change requirements.

Error A1062 Description. One of the system modules in which the enhancement change was implemented has two processing modes for data. These two modes are listed in the user manual. When run in one of the two possible processing modes, the enhancement generated a set of errors, which were put under the heading A1062. At the phase these errors were found, the enhancement had already successfully passed the tests for the other processing mode. The maintainer should have designed a solution to handle both modes correctly.

Error origin. Change design analysis.

Error domain. Lack of application domain knowledge.

Maintenance process flaw(s).

- Personnel-related: lack of experience and/or training with respect to the application domain.
- Other: none noted.

4.5. Recommendations

4.5.1. Lessons learned

Based on the results from Step 5, further complementary investigations (e.g., measurement-based), related to specific issues that have not been fully resolved by the qualitative analysis process, should be identified. Moreover, a first set of suggestions for maintenance process improvement should be devised.

The lessons learned are classified according to the taxonomy of maintenance flaws defined in Figure 8. By performing an overall analysis of the change causal analysis results (Step 6), we abstracted a set of issues detailed in the following subsections.

4.5.2. Organization

Two lessons were learned in the organizational area:

(1) There is a large communication cost overhead between maintainers and users, e.g., release standard documentation, meetings and management forms. In an effort to improve the communication between all the participants of the maintenance process, non-technical, communication-orientated activities have been emphasized. At first glance, this seems to represent about 40% (rough approximation) of the maintenance effort. This figure seems excessive, especially when considering the apparent communication problems (next paragraph).

(2) Despite the number of release meetings and documents, disagreements and misunderstandings seem to disturb the maintenance process until late in the release cycle. For example, design issues that should be settled at the end of the release design review (RDR) meeting keep emerging until acceptance testing is completed.

As a result, it seems that the administrative process and organization scheme should be investigated in order to optimize communication and sign-off procedures, especially between users and maintainers.

4.5.3. Process

Four lessons were learned in the process area:

(3) The tools and techniques used have been developed by maintainers themselves and do not belong to a standard package provided by the organization. Some *ad hoc* technology transfer seems to take place in order to compensate for the lack of a global, commonly agreed upon strategy.

(4) The task leader has been involved in the maintenance of GTDS for a number of years. His expertise seems to compensate for the lack of system documentation. He is also in charge of the training of new personnel (some of the easy changes are used as an opportunity for training). Thus, the process relies heavily on the expertise of one or two persons.

(5) The fact that no historical database of changes exists makes some changes very difficult. Maintainers very often do not understand the semantics of a piece of code added in a previous correction. This seems to be partly due to emergency patching for a mission which was not controlled and cleaned up afterwards (this has recently been addressed), personnel turnover, and a lack of written requirements with respect to performance, precision and platform configuration constraints.

(6) For many of the complex changes, requirements are often ambiguous and incomplete, from a maintainer's perspective. As a consequence, requirements are often unstable until very late in the release process. While prototyping might be necessary for some of them, the users and maintainers do not recognize it as such. Moreover, there is no well-defined standard for expressing change requirements in a style suitable for both maintainers and users.

4.5.4. Product

Three lessons were learned in the product area:

(7) System documentation other than the user's guide is not fully maintained and not trusted by maintainers. Source code is currently the only reliable source of information used by maintainers.

(8) GTDS has a large number of users. As a consequence, the requirements of this system are varied with respect to the hardware configurations on which the system must be able to run, the performance and precision needs, etc. However, no requirement analysis document is available and maintained in order to help the maintainers devise optimal change solutions.

(9) Because of budget constraints, there is no document reliably defining the hardware and precision requirements of the system. Considering the large number of users and platforms on which the system runs, and the rapid evolution of users' needs, this would appear necessary in order to avoid confusion while implementing changes.

4.5.5. People

Two lessons were learned in the people area:

(10) There is a lack of understanding of operational needs and constraints by maintai-

ners. Release meetings were supposed to address such issues but they seem to be inadequate in their current form.

(11) Users are mainly driven by short-term objectives that are aimed at satisfying particular mission requirements. As a consequence, there is a very limited long-term strategy and budget for preventive maintenance. Moreover, the long-term evolution of the system is not driven by a well-defined strategy and maintenance priorities are not clearly identified.

4.5.6. General recommendations

As a general set of recommendations and based on the analysis presented in this paper, we suggested the following set of actions to the GTDS maintenance project as three additional lessons learned:

(12) A standard (that may simply contain guidelines and checklists) should be set up for defining and documenting change requirements. Both users and maintainers should give their input with respect to the content of this standard since it is intended to help them communicate with each other.

(13) The conformance to the defined release process should be improved, e.g., through team building and training. In other words, the release documents and meetings should more effectively play their specified role in the process, e.g., the RDR meeting should settle all design disagreements and inconsistencies.

(14) Those parts of the system that are highly convoluted as a result of numerous modifications should be redesigned and documented for more productive and reliable maintenance. Technical task leaders should be able to point out the sensitive system units.

5. REFLECTIONS ON Q-MOPP AND RELATED WORK

In this section we analyse our method, which applies qualitative analysis methods for the characterization and evaluation of software maintenance. We discuss our method in relation to other related work, and begin by discussing the benefits and drawbacks of our method, Q-MOPP:

- Benefits: it provides us with context specific and rich information, which in turn enables tailored evaluation. As a consequence, results are easy to use as a basis for taking improvement action. In addition, each project contributes to the refinement of existing taxonomies, checklists and guidelines to ease future analyses and evaluations.
- Drawbacks: it is labour-intensive, and needs the co-operation of experts, especially for change analysis. In addition, evaluation results are sometimes difficult to generalize even when they point to changes that may be applicable to other projects in the organization. But by providing certain kinds of information, such as how the evaluation was performed, a meta-analysis of different evaluations may be performed (Cook *et al.*, 1992).

It can also be seen that two levels of improvement exist in our method: improving a

specific project (specific recommendations, such as improving communication), and improving the existing improvement practices (such as redesigning data collection forms). Our evaluation method evolved while performing the case study described in this paper. The initial organizational modelling approach was much simpler, identifying organizational entities and the information flows between them. During the change analysis phase, when we were identifying areas of improvement, we realized we needed a more sophisticated organization model that would capture dependency relationships. In addition, we learned from the case study that the existing weekly maintenance effort data collection form would have to be revised to reflect more accurately the existing maintenance process. The revised forms in turn allowed us to develop a quantitative effort prediction model (Basili *et al.*, 1996).

Our method considers three aspects when developing a descriptive model: organization, product and process. Dart, Christie and Brown (1993) use different groupings: tools, people and process. In our method we consider people to be part of the organization, and tools are considered to be part of the process. McGowan and Bohner (1993) present a process assessment method that makes use of process modelling, but it lacks a clear focus for assessment. When analysing the process, we chose to focus on defects generated during the system release generation process, because these would be defects that would be related to the maintenance process itself. Nakajo and Kume (1991) did another study that relates product defects to process faults. The work by Stark and Oman (1995) focuses on assessing the risks related to a given release by estimating its ‘complexity’. Thus, managers may attempt to alleviate or prevent these risks. Q-MOPP focuses on identifying the problems associated with one or several releases after their completion. In relation to ISO 15504 (SPICE) and the SEI CMM, Q-MOPP is not a competing approach, but rather a specific method that can be used within the context of a more general model. For example, Q-MOPP could be used by a project as part of an organization’s efforts to implement SEI CMM Level 2 practices.

However, qualitative analysis is a priori limited since it does not allow us to quantify precisely the impact of various organizational, technical and process related factors on maintenance cost and quality. Thus, the planning of the release is sometimes arbitrary, monitoring its progress is extremely difficult and its evaluation remains subjective.

Hence, there is a need for a data collection program for GTDS and across all the maintenance projects of the organization. In order to reach such an objective, we have to base the design of such a measurement program on the results provided by this and similar studies. In addition, we need to model more rigorously the maintenance organization and processes so that precise evaluation criteria can be defined. Preliminary results from the current maintenance measurement program can be found in Basili *et al.* (1996).

6. CONCLUSION

Characterizing and understanding software maintenance processes and organizations are necessary, if effective management decisions are to be made and if adequate resource allocation is to be provided. Also, in order to plan and efficiently organize a measurement program—a necessary step towards process improvement (Basili and Rombach, 1988)—we need to characterize better the maintenance environment and its *specific* problems.

The difficulty of performing such a characterization stems from the fact that the people involved in the maintenance process, who have the necessary information and knowledge, cannot perform it because of their inherently partial perspective on the issues and the tight time constraints of their projects. Therefore, a well-defined characterization and bottom–up assessment process, which is cost-effective, generic but tailorable, reasonably objective and applicable by outsiders, needs to be devised.

In this paper, we have presented such an empirically refined process (referred to as Q-MOPP) which allows an organization to gain an in-depth understanding of the issues involved in its maintenance projects. Q-MOPP is intended to provide, to the extent possible, a well-defined, repeatable and traceable process. However, as for many qualitative analysis approaches, some degree of subjectivity is to be expected.

In our experiences with Q-MOPP, we have been able to gather useful information upon which management and technical decisions could be based regarding the studied maintenance process and organization. Besides the case study presented in this paper, this method was also used to analyse several other maintenance projects in the NASA-SEL in order to better understand project similarities and differences in this environment.

Although this method was applied so far in only one environment, there is no reason to believe that it could not be applicable to other maintenance environments. It is general enough to be tailored and followed in most maintenance organizations. The case study presented in this paper represents an analysis effort of approximately two person-months. Considering that the application of Q-MOPP should become easier over time, we do not expect the cost of such an analysis to exceed a few person-months for any given organization.

Acknowledgements

We are grateful to everyone at Computer Sciences Corporation who was involved in the various case studies on which this paper is based. This work would not have been possible without their collaboration. In particular, we would like to thank Don Squier and Steve Condon.

References

Bandinelli, S., Fuggetta, A., Lavazza, L., Loi, M. and Picco, G. P. (1995) ‘Modelling and improving an industrial software process’, *IEEE Transactions on Software Engineering*, **21**(5), 440–454.

Basili, V. R., Briand, L., Condon, S., Melo, W. L., Seaman, C. and Valett, J. (1996) ‘Understanding and predicting the process of software maintenance releases’, in *Proceedings of the 18th International Conference on Software Engineering*, IEEE Computer Society Press, Los Alamitos CA, pp. 464–474.

Basili, V. R. and Rombach, H. D. (1988) ‘The TAME project: towards improvement-oriented software environments’, *IEEE Transactions on Software Engineering*, **14**(6), 758–773.

Bendifallah, S. and Scacchi, W. (1987) ‘Understanding software maintenance work’, *IEEE Transactions on Software Engineering*, **13**(3), 311–323.

Bennett, K. H., Cornelius, B., Munro, M. and Robson, D. (1991) ‘Software maintenance’, in McDermid, J, (Ed), *Software Engineer’s Reference Book*, Butterworth–Heinemann Ltd., Boston MA, pp. 20/1–20/18.

Briand, L. C., Basili, V. R., Kim, Y.-M. and Squier, D. R. (1994) ‘A change analysis process to characterize software maintenance projects’, in *Proceedings of the International Conference on Software Maintenance*, IEEE Computer Society Press, Los Alamitos CA, pp. 38–49.

Briand, L. C., Melo, W. L., Seaman, C. and Basili, V. R. (1995) 'Characterizing and assessing a large–scale software maintenance organization', in *Proceedings of the 17th International Conference on Software Engineering*, IEEE Computer Society Press, Los Alamitos CA, pp. 133–143.

Chapin, N. (1987) 'The job of software maintenance', in *Proceedings of the Conference on Software Maintenance—1987*, IEEE Computer Society Press, Los Alamitos CA, pp. 4–12.

Condon, S., Valett, J., Briand, L., Kim, Y.-M. and Basili, V. R. (1995) '*Maintenance process of three FDD projects*', NASA/GSFC SEL white paper, available from authors, 19 pp.

Cook, T. D., Cooper, H., Cordray, D., Hartmann, H., Hedges, L., Light, R., Louis, T. and Mosteller, F. (1992) *Meta-analysis for Explanation: A Casebook*, Russell Sage Foundation, New York NY, 378 pp.

Curtis, B., Krasner, H. and Iscoe, N. (1988) 'A field study of the software design process for large systems', *Communications of the ACM*, **31**(11), 1268–1287.

Curtis, B., Kellner, M. I. and Over, J. (1992) 'Process modelling', *Communications of the ACM*, **35**(9), 75–90.

Dart, S., Christie, A. M. and Brown, A. W. (1993) '*A case study in software maintenance*', Technical Report CMU/SEI-93-TR-8 or ESC-TR-93-185, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA, 50 pp.

Gilgun, J. F. (1992) 'Definitions, methodologies, and methods in qualitative family research', in Gilgun, J. F., Daly, K. and Handel, G. (Eds), *Qualitative Methods in Family Research*, Sage Publications, Newbury Park CA, pp. 37–61.

Harjani, D.-R. and Queille, J.-P. (1992) 'A process model for the maintenance of large space systems software', in *Proceedings of the Conference on Software Maintenance*, IEEE Computer Society Press, Los Alamitos CA, pp. 127–136.

Hariza, M., Voidrot, J. F., Minor, E., Pofelski, L. and Blazy, S. (1992) 'Software maintenance: an analysis of industrial needs and constraints', in *Proceedings of the Conference on Software Maintenance*, IEEE Computer Society Press, Los Alamitos CA, pp. 18–26.

Madhavji, N., Höltje, D., Hong, W. and Bruckhaus, T. (1994) 'Elicit: a method for eliciting process models', in *Proceedings of the 3rd International Conference on Software Process,* IEEE Computer Society Press, Los Alamitos CA, pp. 112–122.

McGowan, C. L. and Bohner, S. A. (1993) 'Model based process assessments', in *Proceedings of the 15th International Conference on Software Engineering*, IEEE Computer Society Press, Los Alamitos CA, pp. 202–211.

Miles, M. B. and Huberman, A. M. (1994) *Qualitative Data Analysis: An Expanded Sourcebook,* Sage Publications, Thousand Oaks CA, 338 pp.

Nakajo, T. and Kume, H. (1991) 'A case history analysis of software error cause–effect relationships', *IEEE Transactions on Software Engineering*, **17**(8), 830–838.

Patton, M. Q. (1990) *Qualitative Evaluation and Research Methods*, 2nd edn., Sage Publications, Newbury Park CA, 532 pp.

Rombach, H. D. and Ulery, B. T. (1989) 'Establishing a measurement based maintenance improvement program: lessons learned in the SEL', in *Proceedings of the Conference on Software Maintenance—1989*, IEEE Computer Society Press, Los Alamitos CA, pp. 50–57.

Rombach, H. D., Ulery, B. T. and Valett, J. D. (1992) 'Toward full cycle control: adding maintenance measurement to the SEL', *Journal of Systems and Software*, **18**(2), 125–138.

Swanson, E. B. and Beath, C. M. (1988) 'The use of case study data in software management research', *Journal of Systems and Software*, **8**(1), 63–71.

Stark, G. E. and Oman, P. W. (1995) 'A survey instrument for understanding the complexity of software maintenance', *Journal of Software Maintenance*, **7**(6), 421–441.

Shelly, A. and Sibert, E. (1992) 'Qualitative analysis: a cyclical process assisted by computer', in *Qualitative Analysis* series, Oldenbourg Verlag, Munich, pp. 71–114.

Wolcott, H. F. (1994) *Transforming Qualitative Data: Description, Analysis, and Interpretation,* Sage Publications, Thousand Oaks CA, 433 pp.

Yu, E. and Mylopoulos, J. (1994) 'Understanding "why" in software process modelling, analysis, and design', in *Proceedings of the 16th International Conference on Software Engineering*, IEEE Computer Society Press, Los Alamitos CA, pp. 159–168.

Authors' biographies:

Lionel Briand is the head of the Quality and Process Engineering Department at the Fraunhofer Institute for Experimental Software Engineering (FhG IESE), in Germany. Lionel started as a software engineer at CISI Ingénierie, France. He later joined the NASA Software Engineering Laboratory (SEL), a research consortium of NASA Goddard Space Flight Center (GSFC), the University of Maryland and Computer Sciences Corporation. He also served as the lead researcher in software engineering at the Computer Research Institute of Montreal (CRIM) in Canada. Lionel earned, with high honours, a Ph.D degree in Computer Science from the University of Paris XI, France. email: briand@iese.fhg.de

Yong-Mi Kim works at Q-Labs on software process improvement and technology transfer for software using organizations. She previously worked on NASA Goddard Space Flight Center Software Engineering Laboratory research projects in software maintenance. Her work focused on applying qualitative research methods to software maintenance as a way of complementing the quantitative data already available. Yong-Mi received a B.S. degree in Computer Science from Virginia Polytechnic Institute and State University in Blacksburg Virginia, and an M.S. degree in Computer Science from the University of North Carolina in Chapel Hill. email: ymk@q-labs.com

Walcélio L. Melo is the Object-oriented Technologies and Network Computing Architecture Practice Leader at Oracle Brazil Government Vertical, and a Professor at Católica University of Brasília. Before joining Oracle Brazil, he had been the Software Engineering Lead Researcher at CRIM and an Adjunct Professor at McGill University in Montréal, Canada. Walcélio has also been a Faculty Researcher Associate at the University of Maryland in College Park, and a member of the NASA Software Engineering Laboratory at Goddard Space Flight Center. He holds a Ph.D. degree in Computer Science from the University of Grenoble in France. email: wmelo@br.oracle.com

Carolyn Seaman is a faculty researcher in the Experimental Software Engineering Group at the University of Maryland in College Park. Her research includes such topics as the organization structure of development groups, commercial off-the-shelf (COTS) software use in development and maintenance, technology transfer and software maintenance. She has also worked as a software developer and maintainer in a number of different environments. Carolyn earned her B.A. from the College of Wooster (Ohio), her M.S. from Georgia Institute of Technology and her Ph.D. from the University of Maryland in College Park. email: cseaman@cs.umd.edu

Victor R. Basili is a Professor of Computer Science at the University of Maryland in College Park. He is a founder and director of the Software Engineering Laboratory at the NASA Goddard Space Flight Center. Victor has been very active in professional associations, has received awards recognizing his contributions, and is a Fellow of the IEEE and of the ACM. He holds a B.S. from Fordham University in New York, an M.S. from Syracuse University in New York and a Ph.D. from the University of Texas at Austin. email: basili@cs.umd.edu

